Operating Systems - Assignment

COMP2006

Jordan Yeo
17727626

Curtin University
Science and Engineering

Contents

Introduction

The following report is for Operating Systems Assignment for 2017. It will detail how mutual exclusion was achieved for processes and threads. Also the processes and threads that accessed shared resources. Submitted alongside this report is a README, source code and test inputs and outputs.

1 Process MSSV

For this implementation of the solution, mutual exclusion is achieved by having the parent (consumer) wait for all child (producer) processes to complete execution before continuing. This waiting is implemented by having the parent wait for the semParent semaphore before acquiring mutex lock to check if all children have finished. Once a child completes execution is waits for the semParent semaphore before acquiring a mutex lock for itself to update a shared resource, indicating it is finished.

$$wait(semParent)$$
$$wait(semMutex)$$
$$// \ critical \ section$$
$$signal(semMutex)$$
$$signal(semParent)$$

To ensure mutual exclusion for the shared resources (buffer1, buffer2 and counter) the child waited to acquire a mutex lock before entering its critical section to modify the shared resources. Since buffer1 was never modified and only read from this did not require a mutex lock to be obtained before reading.

The semaphore required (semMutex and semParent), buffer1, buffer2 and counter were implemented using shared memory. The following POSIX shared memory functions were used to create shared memory, and the corresponding functions to close the shared memory:

$$shm\_link()$$
$$ftruncate()$$
$$mmap()$$
$$shm\_unlink()$$
$$close()$$
$$munmap()$$

Zombie processes were killed with the use of signal(SIGCHILD, SIG_IGN).

2 Thread MSSV

To achieve mutual exclusion in the multi-threaded program, the parent (consumer) must wait for all threads (producer) to complete execution. The parent uses *pthread_lock_mutex()* to lock the mutex then peforms a *pthread_cond_wait()* on a condition variable. The condition variable represents how many threads are still executing. Once the thread completes execution is acquires a mutex lock to alter the condition variable.

In the thread MSSV the shared resources are declared as global variables. Threads have access to the variables declared global in the parent. Before a thread could access the shared resources it

would first need to acquire a mutex lock. The function *pthread_lock_mutex()* blocks the caller if the mutex is in use by another. It can then alter the shared resources, counter and buffer2, before releasing the mutex lock.
Threads

3 Testing

3.1 Method

To test each implementation of MSSV worked as intended multiple input files were used. With these input files, multiple delays were chosen as well. Input files of various 9x9 numbers were used. Input files were used that were valid, contained one error, contained multiple errors. Also tested were smaller grids than 9x9 and empty files.

Testing was performed on the lab machines in various rooms of Building 314, Level 2.

3.2 Errors

There are no known errors in the process MSSV and the thread MSSV. Care has been taken to ensure potential memory leaks are prevented, by using the appropriate measures to free allocated memory. Memory leaks are not present in the testing of each MSSV currently performed.

3.3 Input Files

$$specTest = \begin{matrix} 6 & 2 & 4 & 5 & 3 & 9 & 1 & 8 & 7 \\ 5 & 1 & 9 & 7 & 2 & 8 & 6 & 3 & 4 \\ 8 & 3 & 7 & 6 & 1 & 4 & 2 & 9 & 5 \\ 1 & 4 & 3 & 8 & 6 & 5 & 7 & 2 & 9 \\ 9 & 5 & 8 & 2 & 4 & 7 & 3 & 6 & 1 \\ 7 & 6 & 2 & 3 & 9 & 1 & 4 & 5 & 8 \\ 3 & 7 & 1 & 9 & 5 & 6 & 8 & 4 & 2 \\ 4 & 9 & 6 & 1 & 8 & 2 & 5 & 7 & 3 \\ 2 & 8 & 5 & 4 & 7 & 3 & 9 & 1 & 6 \end{matrix}$$

$$testFail = \begin{matrix} 2 & 2 & 4 & 5 & 3 & 9 & 1 & 8 & 7 \\ 5 & 1 & 9 & 7 & 2 & 8 & 6 & 3 & 4 \\ 8 & 3 & 7 & 6 & 1 & 4 & 2 & 9 & 5 \\ 1 & 4 & 3 & 8 & 6 & 5 & 7 & 2 & 9 \\ 9 & 5 & 8 & 2 & 4 & 7 & 3 & 6 & 1 \\ 7 & 6 & 2 & 3 & 9 & 1 & 4 & 5 & 8 \\ 3 & 7 & 1 & 9 & 5 & 6 & 8 & 4 & 2 \\ 4 & 9 & 6 & 1 & 8 & 2 & 5 & 7 & 3 \\ 2 & 8 & 5 & 4 & 7 & 3 & 9 & 1 & 6 \end{matrix}$$

$$multiFail = \begin{matrix} 2 & 2 & 4 & 5 & 3 & 9 & 1 & 8 & 7 \\ 5 & 1 & 9 & 7 & 2 & 8 & 6 & 3 & 4 \\ 8 & 3 & 7 & 6 & 1 & 4 & 2 & 9 & 5 \\ 1 & 4 & 3 & 8 & 6 & 5 & 7 & 2 & 9 \\ 9 & 5 & 8 & 2 & 4 & 7 & 3 & 6 & 3 \\ 7 & 6 & 2 & 3 & 9 & 1 & 4 & 5 & 8 \\ 3 & 7 & 1 & 9 & 5 & 6 & 8 & 4 & 2 \\ 4 & 9 & 6 & 1 & 8 & 2 & 5 & 7 & 3 \\ 2 & 8 & 5 & 4 & 7 & 3 & 9 & 1 & 6 \end{matrix}$$

3.4 Expected Results

specTest

```
row 1 is valid
row 2 is valid
row 3 is valid
row 4 is valid
row 5 is valid
row 6 is valid
row 7 is valid
row 8 is valid
row 9 is valid
9 out of 9 columns are valid
9 out of 9 sub-grids are valid

There are 27 valid sub-grids, and thus the solution is valid
```

testFail

```
row 1 is invalid
row 2 is valid
row 3 is valid
row 4 is valid
row 5 is valid
row 6 is valid
row 7 is valid
row 8 is valid
row 9 is valid
8 out of 9 columns are valid
8 out of 9 sub-grids are valid

There are 24 valid sub-grids, and thus the solution is invalid
```

multiFail

```
row 1 is invalid
row 2 is valid
row 3 is valid
row 4 is valid
row 5 is invalid
row 6 is valid
row 7 is valid
row 8 is valid
row 9 is valid
7 out of 9 columns are valid
7 out of 9 sub-grids are valid

There are 21 valid sub-grids, and thus the solution is invalid
```

3.5 Actual Results

Figure 1: Processes: specTest

```
[17727626@lab221-c01 partA]$ ./mssv ../testFiles/specTest.txt 10
Validation result from process ID-19029: row 1 is valid
Validation result from process ID-19030: row 2 is valid
Validation result from process ID-19031: row 3 is valid
Validation result from process ID-19032: row 4 is valid
Validation result from process ID-19033: row 5 is valid
Validation result from process ID-19034: row 6 is valid
Validation result from process ID-19035: row 7 is valid
Validation result from process ID-19036: row 8 is valid
Validation result from process ID-19037: row 9 is valid
Validation result from process ID-19038: 9 out of 9 columns are valid
Validation result from process ID-19039: 9 out of 9 sub-grids are valid
There are 27 valid sub-grids, and thus the solution is valid
```

Figure 2: Threads: specTest

```
[17727626@lab221-b01 partB]$ ./mssv ../testFiles/specTest.txt 10
Validation result from thread ID-2979485440: row 1 is valid
Validation result from thread ID-2971092736: row 2 is valid
Validation result from thread ID-2962700032: row 3 is valid
Validation result from thread ID-2954307328: row 4 is valid
Validation result from thread ID-2945914624: row 5 is valid
Validation result from thread ID-2937521920: row 6 is valid
Validation result from thread ID-2929129216: row 7 is valid
Validation result from thread ID-2920736512: row 8 is valid
Validation result from thread ID-2912343808: row 9 is valid
Validation result from thread ID-2903951104: 9 out of 9 columns are valid
Validation result from thread ID-2895558400: 9 out of 9 sub-grids are valid
There are 27 valid sub-grids, and thus the solution is valid
```

Figure 3: Processes: testFail

```
[17727626@lab221-c01 partA]$ ./mssv ../testFiles/testFail.txt 10
Validation result from process ID-19053: row 1 is invalid
Validation result from process ID-19054: row 2 is valid
Validation result from process ID-19055: row 3 is valid
Validation result from process ID-19056: row 4 is valid
Validation result from process ID-19057: row 5 is valid
Validation result from process ID-19058: row 6 is valid
Validation result from process ID-19059: row 7 is valid
Validation result from process ID-19060: row 8 is valid
Validation result from process ID-19061: row 9 is valid
Validation result from process ID-19062: 8 out of 9 columns are valid
Validation result from process ID-19063: 8 out of 9 sub-grids are valid
There are 24 valid sub-grids, and thus the solution is invalid
```

Figure 4: Threads: testFail

```
[17727626@lab221-b01 partB]$ ./mssv ../testFiles/testFail.txt 10
Validation result from thread ID-2959283968: row 1 is invalid
Validation result from thread ID-2950891264: row 2 is valid
Validation result from thread ID-2942498560: row 3 is valid
Validation result from thread ID-2934105856: row 4 is valid
Validation result from thread ID-2925713152: row 5 is valid
Validation result from thread ID-2917320448: row 6 is valid
Validation result from thread ID-2908927744: row 7 is valid
Validation result from thread ID-2900535040: row 8 is valid
Validation result from thread ID-2892142336: row 9 is valid
Validation result from thread ID-2883749632: 8 out of 9 columns are valid
Validation result from thread ID-2875356928: 8 out of 9 sub-grids are valid
There are 24 valid sub-grids, and thus the solution is invalid
```

Figure 5: Processes: multiFail

```
[17727626@lab221-c01 partA]$ ./mssv ../testFiles/multiFail.txt 10
Validation result from process ID-19070: row 1 is invalid
Validation result from process ID-19071: row 2 is valid
Validation result from process ID-19072: row 3 is valid
Validation result from process ID-19073: row 4 is valid
Validation result from process ID-19074: row 5 is invalid
Validation result from process ID-19075: row 6 is valid
Validation result from process ID-19076: row 7 is valid
Validation result from process ID-19077: row 8 is valid
Validation result from process ID-19078: row 9 is valid
Validation result from process ID-19079: 7 out of 9 columns are valid
Validation result from process ID-19080: 7 out of 9 sub-grids are valid
There are 21 valid sub-grids, and thus the solution is invalid
```

Figure 6: Threads: multiFail

```
[17727626@lab221-b01 partB]$ ./mssv ../testFiles/multiFail.txt 10
Validation result from thread ID-3998418688: row 1 is invalid
Validation result from thread ID-3990025984: row 2 is valid
Validation result from thread ID-3981633280: row 3 is valid
Validation result from thread ID-3973240576: row 4 is valid
Validation result from thread ID-3964847872: row 5 is invalid
Validation result from thread ID-3956455168: row 6 is valid
Validation result from thread ID-3990025984: row 7 is valid
Validation result from thread ID-3813844736: row 8 is valid
Validation result from thread ID-3948062464: row 9 is valid
Validation result from thread ID-3939669760: 7 out of 9 columns are valid
Validation result from thread ID-3931277056: 7 out of 9 sub-grids are valid
There are 21 valid sub-grids, and thus the solution is invalid
```

4 README

4.1 Purpose

The program validates an input file containing a sudoku solution. There are two versions. One utilising processes and the other utilising threads.

4.2 Running the Program

To compile the C files into an executable format.

```
make
```

To run the program

```
make run
```

This will only let you run with the preset parameters and they will need to be altered in the Makefile to test other *input files* and *delays*

```
INPUT: ../testFiles/specTest.txt
DELAY: 10
```

Between each time the program is run, the following command should be entered and executed. This is to delete the logfile produced from an invalid test file.

```
make clean
```

4.3 Files

partA

- Makefile
- mssv.c
- mssv.h

partB

- Makefile
- mssv.c
- mssv.h

testFiles

- test files

References

[1] *Interprocess communication using POSIX Shared Memory in Linux*. URL: https://www.softprayog.in/programming/interprocess-communication-using-posix-shared-memory-in-linux.

[2] *POSIX Semaphores*. URL: https://www.softprayog.in/programming/posix-semaphores.

[3] *POSIX Threads Programming in C*. URL: https://www.softprayog.in/programming/posix-threads-programming-in-c.

[4] *POSIX Threads Synchronization in C*. URL: https://www.softprayog.in/programming/posix-threads-synchronization-in-c.

5 Appendices